Minería de Redes Sociales + Métricas de Desempeño

Autor: Morales Ramos Jose Esau

Descripción general

Este cuaderno presenta, **paso a paso**, un pipeline completo para el **análisis de sentimientos y estructura de interacción** en redes sociales, empleando el dataset `Tweets_sintetico.csv`, inspirado en el clásico conjunto de datos *Twitter US Airline Sentiment*.

El objetivo es mostrar cómo integrar **procesamiento de lenguaje natural (NLP)** con **análisis de redes sociales (SNA)**, permitiendo identificar tanto el **tono emocional** de los mensajes como los **patrones de conexión e influencia** entre usuarios.

Incluye tanto **explicaciones conceptuales** como **código ejecutable**, abarcando desde la preparación de datos hasta la detección de comunidades.

Proceso general

1. **Carga del dataset y Análisis Exploratorio (EDA) mínimo**
   - Inspección inicial de los datos (número de tweets, usuarios, menciones, distribución de sentimientos).
   - Limpieza básica de texto y estructura.
2. **Clasificación de sentimiento**
   - Vectorización con **TF-IDF**.
   - Entrenamiento de un modelo de **Regresión Logística** para etiquetar tweets como *positivos*, *negativos* o *neutros*.
3. **Construcción de la red de menciones**
   - Se genera un grafo donde los **nodos** representan usuarios y las **aristas** reflejan menciones directas (`@usuario`).
   - Se analizan propiedades topológicas básicas: número de nodos, aristas y densidad.
4. **Identificación de influencers**
   - Cálculo de métricas de **centralidad** (PageRank, Betweenness, In/Out-Degree).
   - Identificación de usuarios con mayor alcance o intermediación en la red.
5. **Detección de comunidades y modularidad**
   - Aplicación de algoritmos de detección de comunidades (como Louvain o Girvan–Newman).
   - Medición de la **modularidad** para evaluar la segmentación natural de la red.
6. **Visualización del subgrafo Top-N**
   - Se genera un subgrafo con los usuarios más relevantes (por grado o influencia).
   - Representación gráfica para interpretación visual de la red social.

Notas prácticas

> ⚙️ **Datos locales:** se utiliza el archivo `Tweets_Dataset.csv`.
> 🔒 **Compatibilidad:** evita usar `networkx.info()`. En su lugar, usa `G.number_of_nodes()` y `G.number_of_edges()`.
> 💾 **Rendimiento:** si el hardware es limitado, reduce `max_features` en TF-IDF o usa una muestra (`df.sample(1000)`).
> 🌐 **Sin conexión:** el notebook funciona completamente **offline**, sin requerir acceso a Internet.

En resumen:
Este proyecto combina análisis de texto y redes sociales para obtener una visión integral de cómo los usuarios interactúan y expresan sentimientos en plataformas digitales.
Es una base sólida para extender hacia tareas de **detección de temas, influencia social y análisis de comunidades online**.

1) Carga de datos + EDA mínima

Contexto

En este paso inicial se realiza la **inspección exploratoria del dataset** `Tweets_Dataset.csv`, el cual contiene tweets sintéticos que simulan conversaciones entre usuarios y aerolíneas.
El objetivo es comprender la estructura de los datos antes de aplicar técnicas de minería de texto y análisis de redes.

Conceptos clave

- **EDA (Análisis Exploratorio de Datos):**
  Permite conocer la estructura del dataset —número de filas, columnas, valores faltantes y distribución de clases de sentimiento— para garantizar su calidad antes del modelado.

- **Normalización del texto:**
  Se refiere a la limpieza ligera de las reseñas, eliminando URLs, signos repetidos o espacios innecesarios, pero **manteniendo las menciones** (`@usuario`), ya que serán necesarias para construir la red de interacciones.

Resultados esperados

- **Dimensión del dataset:** número total de tweets y características disponibles.
- **Nulos:** verificación de valores faltantes por columna.
- **Distribución de clases:** proporción de sentimientos (*positivo*, *negativo* o *neutral*), útil para detectar desbalanceos en los datos.

> 💡 Este análisis básico sirve como punto de partida para limpiar, procesar y vectorizar el texto en los siguientes pasos del pipeline.

In [16]:
```
# ============================
# 1) Carga de datos + EDA mínima
# ============================
import pandas as pd

# Cargar dataset actualizado
df = pd.read_csv("Tweets_Dataset.csv")

# Estructura general
print("Filas y columnas:", df.shape)
print("\nColumnas disponibles:\n", df.columns.tolist())

# Vista previa
display(df.head(5))

# Nulos
print("\nValores nulos por columna:")
print(df.isna().sum())

# Distribución de clases
print("\nDistribución de sentimientos (%):")
print(df["airline_sentiment"].value_counts(normalize=True).round(3) * 100)
```

```
Filas y columnas: (7000, 15)

Columnas disponibles:
 ['tweet_id', 'airline_sentiment', 'airline_sentiment_confidence', 'negativereason', 'negativereason_con
fidence', 'airline', 'airline_sentiment_gold', 'name', 'negativereason_gold', 'retweet_count', 'text',
'tweet_coord', 'tweet_created', 'tweet_location', 'user_timezone']
```

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airli
0	1	negative	0.630	lost_luggage	0.821	JetB
1	2	neutral	0.917	NaN	0.793	American
2	3	negative	0.854	booking_problems	0.948	American
3	4	neutral	0.463	NaN	0.826	Unit
4	5	positive	0.506	NaN	0.504	American

```
Valores nulos por columna:
tweet_id                          0
airline_sentiment                 0
airline_sentiment_confidence      0
negativereason                 3164
negativereason_confidence         0
airline                           0
airline_sentiment_gold         7000
name                              0
negativereason_gold            7000
retweet_count                     0
text                              0
tweet_coord                    7000
tweet_created                     0
tweet_location                    0
user_timezone                     0
dtype: int64

Distribución de sentimientos (%):
airline_sentiment
negative    48.9
neutral     29.4
positive    21.7
Name: proportion, dtype: float64
```

2) (opcional) Clasificación de sentimiento — TF-IDF + Regresión Logística

Contexto

En esta etapa se implementa un modelo de **análisis de sentimiento supervisado**, con el objetivo de clasificar los tweets en tres categorías:

positivo, **neutral** y **negativo**.

Se aplica una combinación de **TF-IDF (Term Frequency–Inverse Document Frequency)** para representar el texto en forma numérica y **Regresión Logística Multiclase** como modelo lineal de clasificación.

Conceptos clave

- **TF-IDF (Term Frequency–Inverse Document Frequency):**
  Convierte los textos en vectores numéricos, asignando mayor peso a las palabras más representativas y menos a las comunes.
  Permite capturar patrones lingüísticos útiles sin necesidad de embeddings complejos.
- **Regresión Logística Multiclase:**
  Modelo supervisado lineal que estima la probabilidad de pertenencia de un texto a cada clase de sentimiento (`negative` , `neutral` , `positive`).
- **Métricas de evaluación:**
  - *Precisión (Precision):* proporción de predicciones correctas sobre el total de positivas detectadas.
  - *Exhaustividad (Recall):* porcentaje de verdaderos positivos correctamente identificados.
  - *F1-Score:* promedio armónico entre precisión y recall, útil en datasets desbalanceados.
  - *F1-macro / ROC-AUC macro:* promueven una visión general del rendimiento del modelo en todas las clases.

Nota de rendimiento

> Si tu hardware tiene recursos limitados:
>
> - Reduce el parámetro `max_features` en **TF-IDF**, o
> - Usa `ngram_range=(1,1)` para ignorar bigramas y acelerar el procesamiento.

Resumen de la implementación

1. **Carga y limpieza ligera del texto:**
   Se eliminan URLs, signos repetidos y espacios innecesarios, **conservando menciones (`@usuario`)** para el análisis de red posterior.
2. **Vectorización con TF-IDF:**
   Cada tweet se transforma en un vector numérico según la frecuencia relativa de sus palabras.
3. **Entrenamiento del modelo:**
   Se entrena un clasificador **LogisticRegression** con validación estratificada (`train_test_split`) para garantizar equilibrio entre clases.
4. **Evaluación de desempeño:**
   Se generan reportes de métricas (`classification_report` , `confusion_matrix`) y curvas ROC-AUC para visualizar la efectividad del modelo.

In [17]:
```
# ============================
# 2) Clasificación de sentimiento (adaptado a Tweets_Dataset.csv)
# ============================
import re
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
```

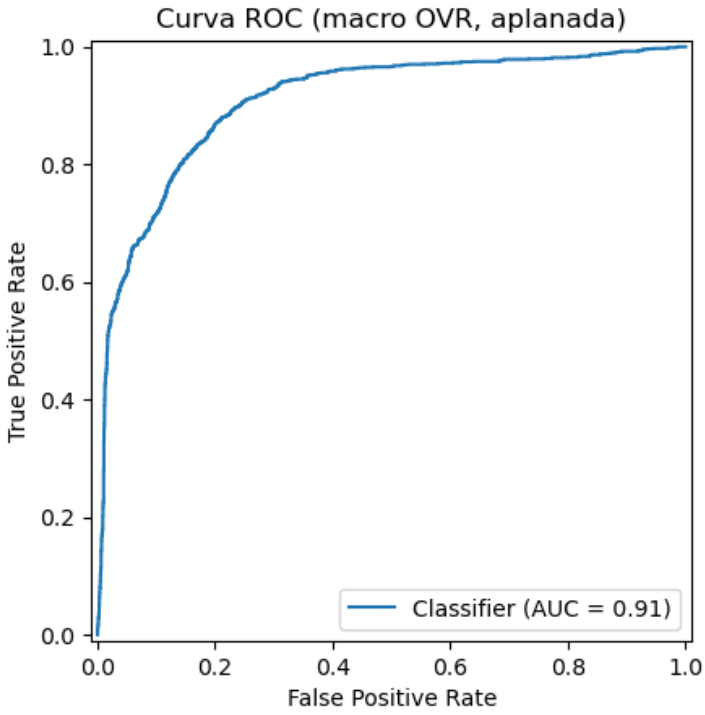
```
    classification_report, confusion_matrix, f1_score, roc_auc_score, RocCurveDisplay
)
from sklearn.preprocessing import label_binarize

# --- Carga ---
df = pd.read_csv("Tweets_Dataset.csv")

# --- Limpieza ligera para 'text_clean' (conservar menciones) ---
def clean_text_keep_mentions(s: str) -> str:
    s = str(s)
    s = re.sub(r"http\S+|www\.\S+", " ", s)                # quitar URLs
    s = re.sub(r"[^\S\r\n]+", " ", s)                      # espacios múltiples
    s = re.sub(r"([!?.,])\1+", r"\1", s)                   # signos repetidos
    # conservar @usuarios; quitar símbolos extraños excepto # y @
    s = re.sub(r"[^A-Za-z0-9@#_\-\s]", " ", s)
    return s.strip().lower()

if "text_clean" not in df.columns:
    df["text_clean"] = df["text"].astype(str).apply(clean_text_keep_mentions)

# --- Split ---
X = df["text_clean"].values
y = df["airline_sentiment"].values
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=42, stratify=y
)

# --- TF-IDF ---
tfidf = TfidfVectorizer(
    max_features=30000,         # baja si tu hardware es limitado
    ngram_range=(1,2),
    min_df=2,
    lowercase=True
)
Xtr = tfidf.fit_transform(X_train)
Xte = tfidf.transform(X_test)

# --- Modelo baseline ---
clf = LogisticRegression(max_iter=200, solver="lbfgs", multi_class="ovr")
clf.fit(Xtr, y_train)

y_pred = clf.predict(Xte)
y_prob = clf.predict_proba(Xte)

print("=== Classification report ===")
print(classification_report(y_test, y_pred, digits=3))

# --- F1 macro ---
f1_macro = f1_score(y_test, y_pred, average="macro")
print("F1 macro:", round(f1_macro, 4))

# --- ROC-AUC macro (OVR) ---
classes = clf.classes_
y_test_bin = label_binarize(y_test, classes=classes)
auc_macro = roc_auc_score(y_test_bin, y_prob, average="macro", multi_class="ovr")
print("ROC-AUC macro (OVR):", round(auc_macro, 4))

# --- Matriz de confusión ---
cm = confusion_matrix(y_test, y_pred, labels=classes)
cm_df = pd.DataFrame(cm, index=[f"true_{c}" for c in classes], columns=[f"pred_{c}" for c in classes])
display(cm_df)

# --- Curva ROC macro (aplanado OVR) ---
plt.figure()
RocCurveDisplay.from_predictions(y_test_bin.ravel(), y_prob.ravel())
plt.title("Curva ROC (macro OVR, aplanada)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```

```
=== Classification report ===
              precision    recall  f1-score   support

    negative      0.759     0.955     0.846       685
     neutral      0.703     0.614     0.655       412
    positive      0.815     0.479     0.603       303

    accuracy                          0.751      1400
   macro avg      0.759     0.682     0.701      1400
weighted avg      0.754     0.751     0.737      1400

F1 macro: 0.7013
ROC-AUC macro (OVR): 0.8912
```

```
C:\Users\Bersd\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:1256: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. Use OneVsRestClassifier(LogisticRegression(..)) instead. Leave it to its default value to avoid this warning.
  warnings.warn(
```

	pred_negative	pred_neutral	pred_positive
true_negative	654	23	8
true_neutral	134	253	25
true_positive	74	84	145

```
<Figure size 640x480 with 0 Axes>
```



3) Análisis de red social sobre menciones (autor → mencionado)

Contexto

En esta etapa se transforma la información de menciones (`@usuario`) dentro de los tweets en una **red social dirigida**, donde:

- **Cada nodo** representa a un usuario de Twitter.
- **Cada arista (flecha)** indica una mención de un autor hacia otro (autor → mencionado).

Ejemplo:
Si `@ana_92` escribe *"Gracias por la ayuda @luisperez"*, se genera una conexión:
→ ana_92 → luisperez

Conceptos clave

- **Grafo dirigido:** los usuarios son nodos y las menciones son aristas con dirección.
- **Métrica de influencia:** los nodos con mayor *In-Degree* son los más mencionados o influyentes.
- **Más aristas = mayor interacción.**

Objetivo: construir la base del análisis de redes sociales, permitiendo identificar quiénes son los usuarios más activos o con mayor visibilidad en la conversación.

In [18]:
```
# ============================
# 3) Red social de menciones (adaptado a Tweets_Dataset)
# ============================
import pandas as pd
import re
import networkx as nx

# --- Cargar dataset ---
df = pd.read_csv("Tweets_Dataset.csv")

# === Extracción de menciones y autores ===
def extract_mentions(text):
    """
    Extrae las menciones (@usuario) desde el texto original.
    Devuelve una lista sin el símbolo '@'.

    Ejemplo:
        >>> extract_mentions("Vuelo terrible @United @AmericanAir")
        ['United', 'AmericanAir']
    """
    if pd.isna(text):
        return []
    return re.findall(r"@(\w+)", str(text))

# Autor (columna 'name' → 'author')
if "name" in df.columns and df["name"].notna().any():
    df["author"] = df["name"].fillna("").astype(str)
else:
    df["author"] = [f"user_{i}" for i in range(len(df))]

# Extraer menciones
df["mentions"] = df["text"].apply(extract_mentions)

# === Construcción del grafo dirigido ===
edges = []
for author, mlist in zip(df["author"], df["mentions"]):
    for mentioned in mlist:
        if mentioned and author.lower() != mentioned.lower():
            edges.append((author, mentioned.lower()))

G = nx.DiGraph()
G.add_edges_from(edges)  # autor → mencionado

print("Nodos:", G.number_of_nodes(), "Aristas:", G.number_of_edges())
print(f"Ejemplo de aristas: {list(G.edges())[:10]}")
```

```
Nodos: 2484 Aristas: 9700
Ejemplo de aristas: [('u_nozpz5', 'jetbluue'), ('u_nozpz5', 'u_d2ries'), ('u_nozpz5', 'americanair'),
('u_nozpz5', 'u_pwm9t3'), ('u_nozpz5', 'jetblue'), ('u_nozpz5', 'jetblu'), ('u_nozpz5', 'southwestair'),
('u_nozpz5', 'u_tnqrvn'), ('u_nozpz5', 'united'), ('u_nozpz5', 'u_o0kcsc')]
```

In [19]:
```
extract_mentions("Vuelo malo @UnitedAirlines @AmericanAir")
```

Out[19]:
```
['UnitedAirlines', 'AmericanAir']
```

In [20]:
```
edges[:5]
```

Out[20]:
```
[('u_nozpz5', 'jetbluue'),
 ('u_nozpz5', 'u_d2ries'),
 ('u_7rmbsm', 'americanair'),
 ('u_7rmbsm', 'u_lf1quy'),
 ('u_oanud9', 'americanair')]
```

Subgrafo: Top 100 nodos por grado total

Contexto

En esta sección se visualiza un **subgrafo** formado por los **100 usuarios más conectados** (con mayor número de menciones totales, tanto enviadas como recibidas).
El objetivo es identificar los principales focos de interacción dentro de la red social.

Conceptos clave

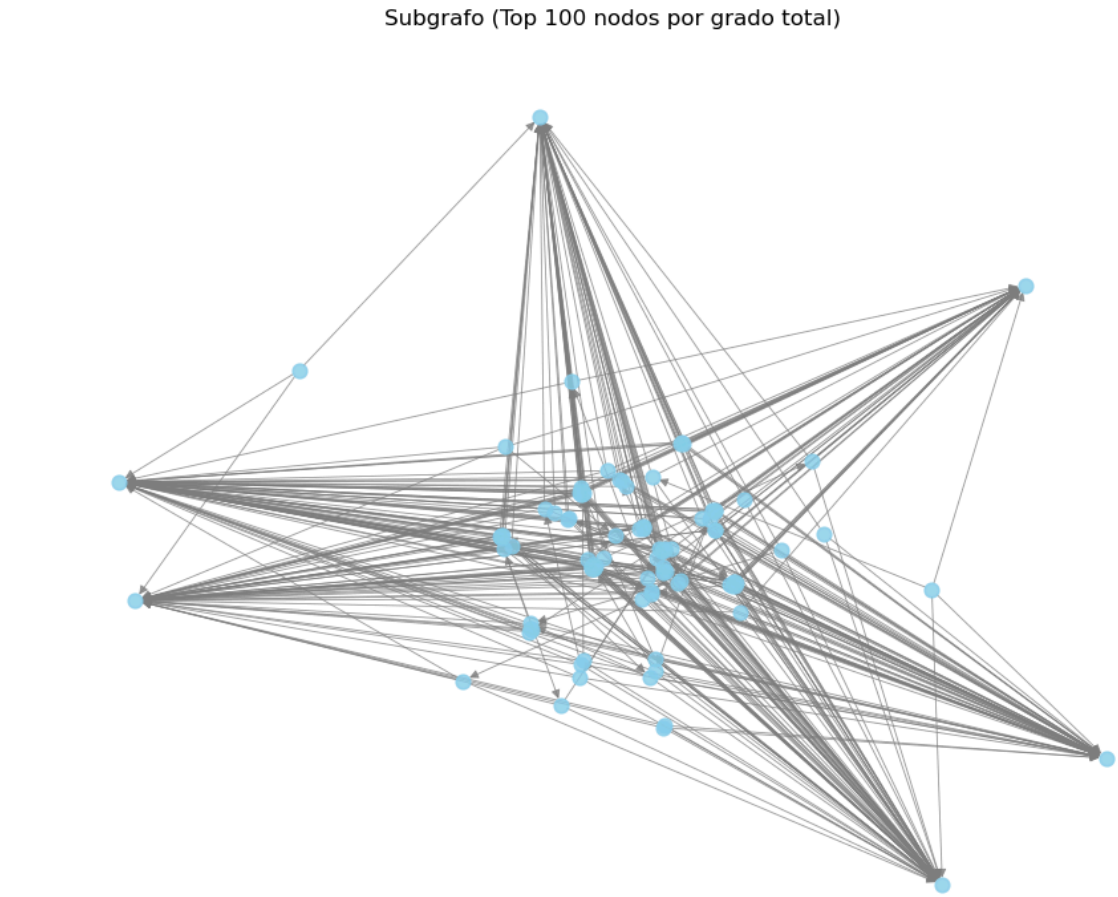
- **Grado total (In + Out):** mide cuántas conexiones tiene un nodo en total.
- Los nodos con **mayor grado** suelen representar usuarios muy activos o populares.
- El **subgrafo** facilita la visualización y reduce la complejidad del grafo completo.

Interpretación:
Este gráfico muestra los usuarios más relevantes en la conversación, permitiendo detectar comunidades o posibles influenciadores dentro del conjunto de tweets.

In [21]:
```
# ============================
# Subgrafo: top 100 nodos por grado (adaptado a Tweets_Dataset)
# ============================
import pandas as pd
import re
import networkx as nx
import matplotlib.pyplot as plt

# --- cargar y construir el grafo si no existe G ---
try:
    G
except NameError:
    df = pd.read_csv("Tweets_Dataset.csv")

    def extract_mentions(text):
        if pd.isna(text):
            return []
        return re.findall(r"@(\w+)", str(text))

    if "name" in df.columns and df["name"].notna().any():
        df["author"] = df["name"].fillna("").astype(str)
    else:
        df["author"] = [f"user_{i}" for i in range(len(df))]

    df["mentions"] = df["text"].apply(extract_mentions)

    edges = []
    for author, mlist in zip(df["author"], df["mentions"]):
        for mentioned in mlist:
            if mentioned and author.lower() != mentioned.lower():
                edges.append((author, mentioned.lower()))

    G = nx.DiGraph()
    G.add_edges_from(edges)

# --- subgrafo con los 100 nodos más conectados (grado total: in+out) ---
degree_dict = dict(G.degree())
top_nodes = [n for n, _ in sorted(degree_dict.items(), key=lambda kv: kv[1], reverse=True)[:100]]
H = G.subgraph(top_nodes).copy()

# --- layout y dibujo ---
pos = nx.spring_layout(H, seed=42, k=None)  # k=None deja a networkx escoger
plt.figure(figsize=(9,7))
nx.draw(
    H, pos,
    node_size=60,
    node_color="skyblue",
```

```
    width=0.6,
    alpha=0.8,
    with_labels=False
)
plt.title("Subgrafo (Top 100 nodos por grado total)")
plt.axis("off")
plt.tight_layout()
plt.show()
```

```
C:\Users\Bersd\AppData\Local\Temp\ipykernel_38920\4134692706.py:55: UserWarning: This figure includes Axes that are not compatible with tight_layout, so results might be incorrect.
  plt.tight_layout()
```



4) Influencers: PageRank + Betweenness

Contexto

Buscamos identificar a los **usuarios más influyentes o centrales** dentro de la red de menciones. Para ello calculamos métricas clásicas de **centralidad**.

Métricas clave

- **PageRank:** importancia "global" de un nodo según quién lo menciona (autoridad).
- **Betweenness:** cuántos caminos pasan por un nodo (actúa como **puente** entre grupos).
- **In-Degree / Out-Degree:** menciones **recibidas** / **realizadas** (popularidad vs. actividad).

Interpretación rápida

- **PageRank alto** ⇒ posibles **influencers** o cuentas de referencia.
- **Betweenness alta** ⇒ **conectores** que enlazan comunidades distintas.
- **In-Degree alto** ⇒ perfiles **muy mencionados**; **Out-Degree alto** ⇒ perfiles muy **activos**.

> Resultado esperado: tabla/ranking con los top-N usuarios por cada métrica para analizar roles y comparar influencia, puente y popularidad.

In [23]:
```
# ============================
# 4) Influencers: PageRank + Betweenness + In/Out-Degree (Tweets_Dataset)
# ============================
import pandas as pd
import networkx as nx

# --- asegurar que el grafo G existe ---
try:
    G
except NameError:
    df = pd.read_csv("Tweets_Dataset.csv")
    import re
    def extract_mentions(text):
        if pd.isna(text): return []
        return re.findall(r"@(\w+)", str(text))
    if "name" in df.columns and df["name"].notna().any():
        df["author"] = df["name"].fillna("").astype(str)
    else:
        df["author"] = [f"user_{i}" for i in range(len(df))]
    df["mentions"] = df["text"].apply(extract_mentions)
    edges = []
    for author, mlist in zip(df["author"], df["mentions"]):
        for mentioned in mlist:
            if mentioned and author.lower() != mentioned.lower():
                edges.append((author, mentioned.lower()))
    G = nx.DiGraph()
    G.add_edges_from(edges)

# === Métricas de centralidad ===

# PageRank (influencia global)
pr = nx.pagerank(G, alpha=0.85)

# Betweenness (puentes entre comunidades)
btw = nx.betweenness_centrality(G, normalized=True)

# In/Out-degree (popularidad/actividad)
indeg = dict(G.in_degree())
outdeg = dict(G.out_degree())

# === Tabla consolidada ===
influ_table = pd.DataFrame({
    "node": list(G.nodes()),
    "pagerank": [pr.get(n, 0.0) for n in G.nodes()],
    "betweenness": [btw.get(n, 0.0) for n in G.nodes()],
    "in_degree": [indeg.get(n, 0) for n in G.nodes()],
    "out_degree": [outdeg.get(n, 0) for n in G.nodes()],
}).sort_values(["pagerank","betweenness","in_degree"], ascending=False)

print("Top 15 influencers (por PageRank, Betweenness e In-degree)")
display(influ_table.head(15))
```

```
Top 15 influencers (por PageRank, Betweenness e In-degree)
```

	node	pagerank	betweenness	in_degree	out_degree
10	united	0.070167	0.000000	1152	0
4	americanair	0.069883	0.000000	1151	0
18	delta	0.048876	0.000000	827	0
15	jetblue	0.044565	0.000000	768	0
24	southwestair	0.041911	0.000000	743	0
31	virginamerica	0.025201	0.000000	481	0
71	uniited	0.000927	0.000000	6	0
228	mericanair	0.000746	0.000000	7	0
1113	aericanair	0.000680	0.000000	7	0
68	u_g7l45r	0.000627	0.016953	7	6
2259	deltta	0.000616	0.000000	3	0
2261	dlta	0.000607	0.000000	2	0
1014	outhwestair	0.000602	0.000000	5	0
1352	nited	0.000587	0.000000	8	0
2324	aamericanair	0.000587	0.000000	3	0

5) Comunidades y modularidad (grafo no dirigido)

Contexto

En este paso se analizan **grupos naturales de usuarios** dentro de la red, detectando comunidades que interactúan más entre sí que con otros.
Ejemplos:

- Seguidores de una aerolínea.
- Usuarios que presentan quejas similares.
- Clientes fieles que mencionan con frecuencia la misma marca.

Conceptos clave

- Se convierte el grafo a **no dirigido** para medir conexiones sin importar quién menciona a quién.
- El algoritmo **Greedy Modularity** identifica comunidades maximizando la **modularidad (Q)**.
- **Q ≈ 0.3–0.5:** comunidades claras.
- **Q > 0.5:** comunidades muy bien definidas.

Interpretación

El número y tamaño de comunidades muestran la **estructura temática** de la red (reclamos, promociones, conversaciones).
Un valor de **Q ≈ 0.4–0.5** indica grupos diferenciados, por ejemplo, usuarios agrupados por aerolínea o tipo de interacción.

In [24]:
```
# ============================
# 5) Comunidades y modularidad (Tweets_Dataset)
# ============================
import networkx as nx
import pandas as pd

# --- asegurar que existe el grafo dirigido G ---
try:
    G
```

```
except NameError:
    df = pd.read_csv("Tweets_Dataset.csv")
    import re
    def extract_mentions(text):
        if pd.isna(text): return []
        return re.findall(r"@(\w+)", str(text))
    if "name" in df.columns and df["name"].notna().any():
        df["author"] = df["name"].fillna("").astype(str)
    else:
        df["author"] = [f"user_{i}" for i in range(len(df))]
    df["mentions"] = df["text"].apply(extract_mentions)
    edges = []
    for author, mlist in zip(df["author"], df["mentions"]):
        for mentioned in mlist:
            if mentioned and author.lower() != mentioned.lower():
                edges.append((author, mentioned.lower()))
    G = nx.DiGraph()
    G.add_edges_from(edges)

# --- convertir a grafo no dirigido ---
Gu = G.to_undirected()

# --- métricas globales ---
density = nx.density(Gu)
avg_clustering = nx.average_clustering(Gu) if Gu.number_of_nodes() > 0 else 0.0

print(f"Densidad: {density:.4f} | Clustering promedio: {avg_clustering:.4f}")

# --- detección de comunidades ---
communities = []
modularity = None
if Gu.number_of_nodes() > 0 and Gu.number_of_edges() > 0:
    comms = list(nx.algorithms.community.greedy_modularity_communities(Gu))
    communities = [sorted(list(c)) for c in comms]
    modularity = nx.algorithms.community.quality.modularity(Gu, comms)

# --- resumen ---
summary_comm = pd.DataFrame({
    "community_id": range(1, len(communities)+1),
    "size": [len(c) for c in communities]
}).sort_values("size", ascending=False)

print("Comunidades detectadas:", len(communities))
print("Modularidad:", round(modularity, 4) if modularity is not None else None)
display(summary_comm.head(10))
```

```
Densidad: 0.0031 | Clustering promedio: 0.1967
Comunidades detectadas: 21
Modularidad: 0.294
```

	community_id	size
0	1	337
1	2	332
2	3	318
3	4	303
4	5	298
5	6	239
6	7	92
7	8	61
8	9	60
9	10	54

Sentimiento medio por comunidad

Objetivo: estimar el **tono promedio** (negativo/neutral/positivo) de cada **comunidad** de usuarios detectada en la red de menciones.

Qué hace la celda

1. **Carga/asegura columnas** (`author` , `mentions`) y construye el **grafo** de menciones si aún no existe.
2. Convierte el grafo a **no dirigido** y obtiene **comunidades** con *Greedy Modularity*.
3. Mapea el **sentimiento textual** de cada tweet a valores numéricos:
   `negative → -1` , `neutral → 0` , `positive → 1` .
4. Asigna a cada **autor** su **id de comunidad** y **agrega por comunidad**:
   - `n_tweets` : cantidad de tweets del grupo.
   - `sent_mean` : promedio del sentimiento.
5. Añade una **interpretación rápida**:
   - `> 0.1` → **positivo**
   - `< -0.1` → **negativo**
   - caso contrario → **mixto/neutral**

Salida: una tabla con el **sentimiento medio por comunidad**, útil para comparar grupos (p.ej., comunidades más críticas vs. favorables).

In [25]:
```
# ============================
# Sentimiento medio por comunidad (Tweets_Dataset)
# ============================
import pandas as pd
import numpy as np
import networkx as nx
import re

# --- Cargar y asegurar columnas base ---
df = pd.read_csv("Tweets_Dataset.csv")

def extract_mentions(text):
    if pd.isna(text): return []
    return re.findall(r"@(\w+)", str(text))

if "author" not in df.columns:
    if "name" in df.columns and df["name"].notna().any():
        df["author"] = df["name"].fillna("").astype(str)
    else:
        df["author"] = [f"user_{i}" for i in range(len(df))]

if "mentions" not in df.columns:
    df["mentions"] = df["text"].apply(extract_mentions)

# --- Grafo y comunidades (si no existen) ---
try:
    communities
    Gu
except NameError:
    G = nx.DiGraph()
    edges = []
    for a, mlist in zip(df["author"], df["mentions"]):
        for m in mlist:
            if m and a.lower() != m.lower():
                edges.append((a, m.lower()))
    G.add_edges_from(edges)
    Gu = G.to_undirected()
    comms = list(nx.algorithms.community.greedy_modularity_communities(Gu))
    communities = [sorted(list(c)) for c in comms]

# --- Sentimiento medio por comunidad ---
sent_map = {"negative": -1, "neutral": 0, "positive": 1}
df["sent_num"] = df["airline_sentiment"].map(sent_map).fillna(0)

node_to_comm = {}
for i, cset in enumerate(communities, start=1):
    for n in cset:
```

```
df["author_comm"] = df["author"].map(node_to_comm)

comm_sent = (
    df.dropna(subset=["author_comm"])
      .groupby("author_comm")["sent_num"]
      .agg(["count", "mean"])
      .rename(columns={"count": "n_tweets", "mean": "sent_mean"})
      .sort_index()
)

comm_sent["interpretación"] = np.where(
    comm_sent["sent_mean"] > 0.1, "positivo",
    np.where(comm_sent["sent_mean"] < -0.1, "negativo", "mixto/neutral")
)

print("Sentimiento medio por comunidad (autores):")
comm_sent
```

```
Sentimiento medio por comunidad (autores):
```

Out[25]:

author_comm	n_tweets	sent_mean	interpretación
1	1019	-0.294406	negativo
2	976	-0.254098	negativo
3	956	-0.275105	negativo
4	856	-0.279206	negativo
5	865	-0.299422	negativo
6	699	-0.284692	negativo
7	232	-0.193966	negativo
8	141	-0.304965	negativo
9	152	-0.282895	negativo
10	149	-0.275168	negativo
11	129	-0.224806	negativo
12	109	-0.155963	negativo
13	117	-0.230769	negativo
14	94	-0.297872	negativo
15	88	-0.147727	negativo
16	100	-0.310000	negativo
17	86	-0.186047	negativo
18	81	-0.296296	negativo
19	65	-0.276923	negativo
20	77	-0.272727	negativo
21	9	-0.111111	negativo

In [26]:
```
print("Autores con comunidad asignada:", df["author_comm"].notna().sum(), "de", len(df))
```

```
Autores con comunidad asignada: 7000 de 7000
```

Visualización final de comunidades (Tweets_Dataset)

Objetivo: representar visualmente las **comunidades detectadas** dentro del grafo de menciones, mostrando cómo se agrupan los usuarios en torno a interacciones comunes.

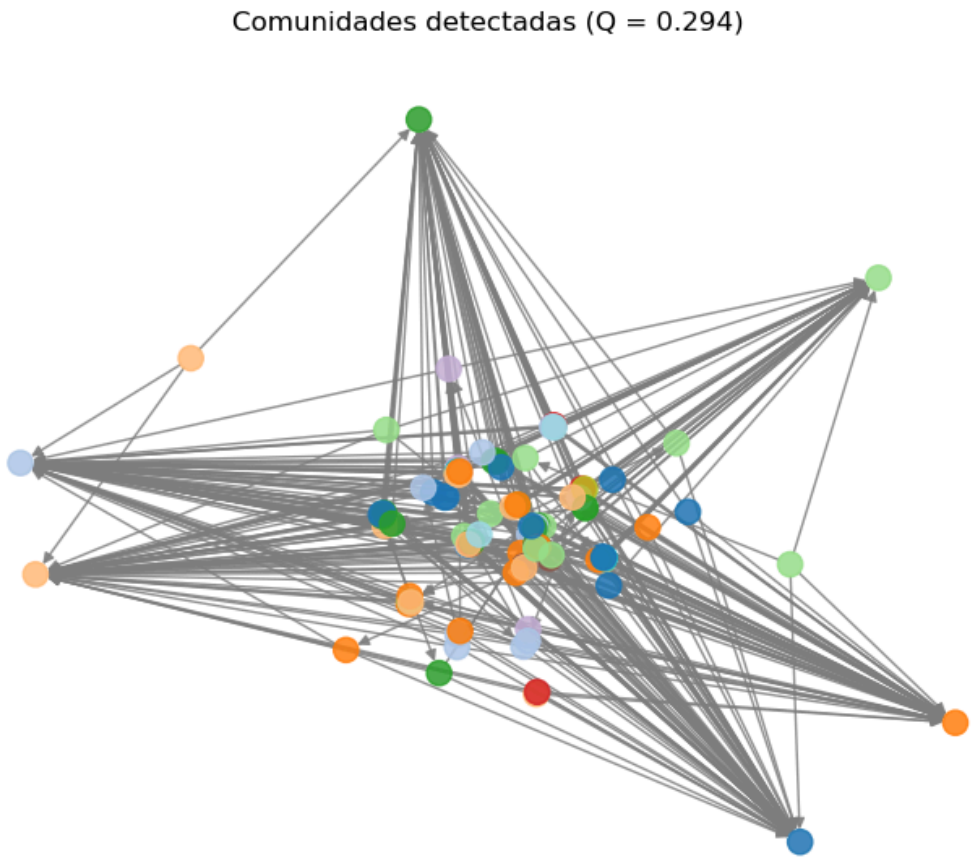
Qué hace la celda

1. **Subgrafo Top-100:** selecciona los 100 nodos más conectados del grafo original (`G`) para facilitar la visualización.
2. **Layout:** calcula las posiciones de los nodos usando `spring_layout` , que distribuye los puntos para minimizar el cruce de aristas.
3. **Colores por comunidad:** asigna un color distinto a cada comunidad detectada por el algoritmo de modularidad.
4. **Gráfico:** dibuja el subgrafo con `networkx` , donde:
   - El color indica la comunidad.
   - El tamaño y densidad de conexiones reflejan la centralidad de los usuarios.

Interpretación

- Los **grupos de color uniforme** representan comunidades de usuarios que interactúan más entre sí.
- Las **zonas con nodos densamente conectados** pueden corresponder a usuarios que discuten temas similares o mencionan la misma aerolínea.
- El título incluye el valor de **modularidad (Q)**, que mide cuán bien definidas están las comunidades (valores entre **0.3–0.6** indican una estructura clara).

In [27]:
```
# ============================
# Visualización de comunidades (Tweets_Dataset)
# ============================
import matplotlib.pyplot as plt
import networkx as nx

# --- asegurar subgrafo H y posiciones ---
try:
    H, pos
except NameError:
    # si no existen, tomar top 100 nodos más conectados del grafo G
    degree_dict = dict(G.degree())
    top_nodes = [n for n, _ in sorted(degree_dict.items(), key=lambda kv: kv[1], reverse=True)[:100]]
    H = G.subgraph(top_nodes).copy()
    pos = nx.spring_layout(H, seed=42)

# --- asignar color por comunidad ---
color_map = {}
for i, cset in enumerate(communities):
    for node in cset:
        color_map[node] = i

colors = [color_map.get(n, 0) for n in H.nodes()]

# --- graficar ---
plt.figure(figsize=(9, 7))
nx.draw_networkx(
    H, pos,
    node_color=colors,
    cmap=plt.cm.tab20,
    node_size=120,
    edge_color="gray",
    alpha=0.85,
    with_labels=False
)
plt.title(f"Comunidades detectadas (Q = {modularity:.3f})")
plt.axis("off")
plt.show()
```



Visualización ligera (subgrafo Top-50)

Objetivo: mostrar una vista más clara y estética del grafo de comunidades, enfocándose solo en los **50 nodos más relevantes** para destacar las conexiones principales.

Qué hace la celda

1. **Colores por comunidad:** asigna una paleta de colores consistente y fija (usando `tab20`) para identificar visualmente las comunidades.
2. **Tamaño y ancho proporcional:**
   - El **tamaño del nodo** se escala según su *in-degree* (cantidad de menciones recibidas).
   - El **ancho de las aristas** depende del *peso* o frecuencia de conexión.
3. **Dibujo del grafo:** renderiza el subgrafo `H` con colores de comunidad, tamaños proporcionales y una transparencia ajustada para mejorar la lectura.
4. **Leyenda automática:** genera etiquetas indicando la **comunidad y cantidad de miembros** visibles en el subgrafo.

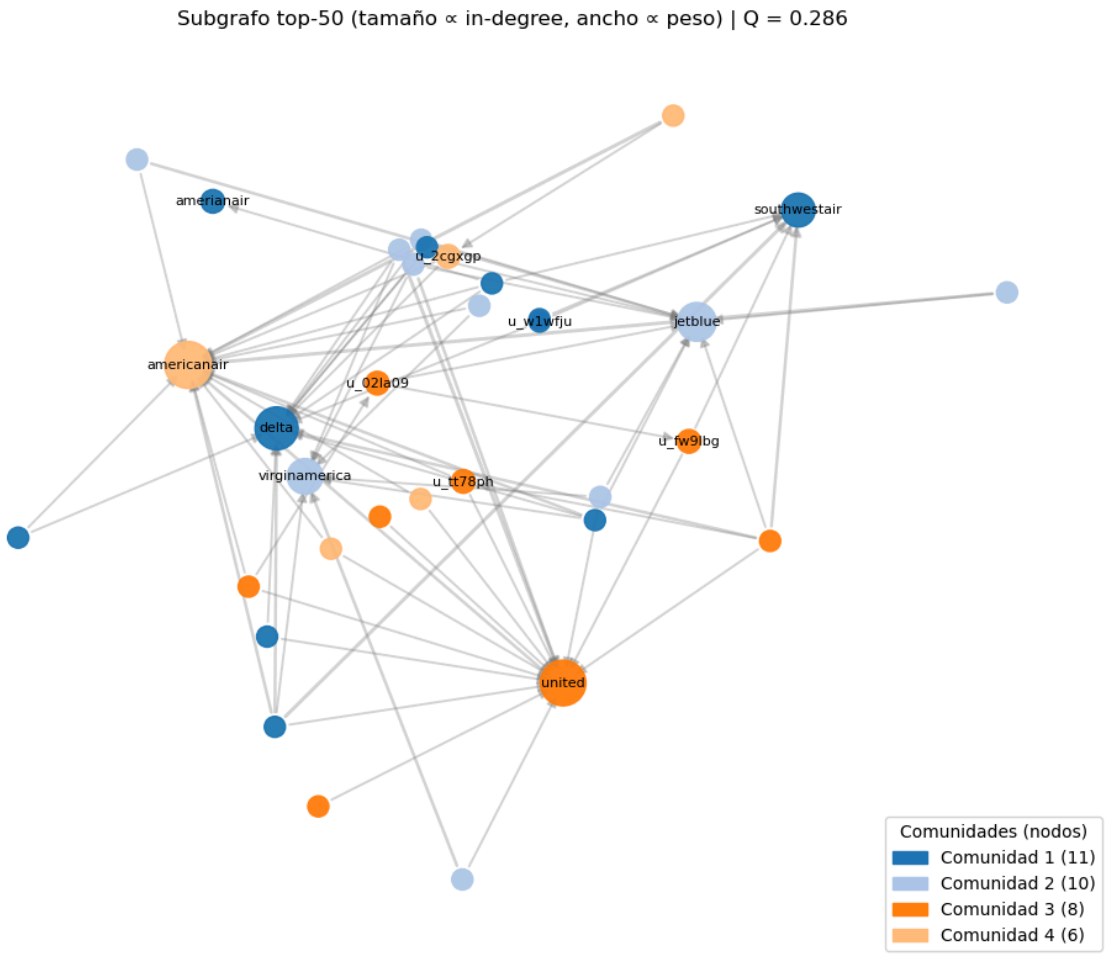
Interpretación

- Los **nodos grandes** representan usuarios más mencionados o influyentes.
- Los **colores** agrupan usuarios pertenecientes a una misma comunidad.
- Las **aristas gruesas** indican interacciones frecuentes entre dos nodos.
- Permite observar la estructura general sin la sobrecarga visual de todo el grafo (ideal para reportes o análisis rápidos).

In [33]:
```
# ============================
# Leyenda y colores CONSISTENTES por comunidad
# ============================
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
```

```
# comms (lista de sets) y H/pos ya creados.
# 1) Asignar ID de comunidad a cada nodo del subgrafo H
comm_id = {}
for i, cset in enumerate(comms):   # i = 0..K-1
    for n in cset:
        comm_id[n] = i

# 2) Paleta fija y mapeo consistente id -> color
K = len(comms)
base_cmap = plt.cm.get_cmap("tab20", max(K, 20))  # asegura ≥20 colores
palette = [base_cmap(i) for i in range(K)]        # lista de K colores
color_for_cid = {cid: palette[cid % len(palette)] for cid in range(K)}

# 3) Colores de nodos usando el MISMO mapeo
node_colors = [color_for_cid.get(comm_id.get(n, -1), (0.7,0.7,0.7,1.0)) for n in H.nodes()]

# 4) Tamaños y anchos (como antes)
in_deg = dict(H.in_degree())
vals = np.array(list(in_deg.values()), dtype=float) if in_deg else np.array([1.0])
rango = np.ptp(vals) if np.ptp(vals) > 0 else 1.0
sizes = 150 + (vals - vals.min())/rango * 600

w = np.array([H[u][v].get("weight",1) for u,v in H.edges()], dtype=float)
ew = 0.6 + np.log1p(w)

# 5) Layout y dibujo
plt.figure(figsize=(10,8))
nx.draw_networkx_edges(H, pos, alpha=0.35, width=ew, edge_color="gray", arrows=True, arrowsize=12)
nx.draw_networkx_nodes(H, pos, node_size=sizes, node_color=node_colors, alpha=0.95)
nx.draw_networkx_labels(H, pos,
                        labels={n:n for n,_ in sorted(in_deg.items(), key=lambda kv: kv[1], reverse=Tru
                        font_size=8)

plt.title(f"Subgrafo top-50 (tamaño ∝ in-degree, ancho ∝ peso) | Q = {modularity:.3f}")
plt.axis("off")

# 6) LEYENDA usando el MISMO mapeo color_for_cid
patches = []
for cid in range(K):
    members = [n for n in H.nodes() if comm_id.get(n, -1) == cid]
    if members:  # solo comunidades presentes en H
        patches.append(mpatches.Patch(color=color_for_cid[cid],
                                      label=f"Comunidad {cid+1} ({len(members)})"))

plt.legend(handles=patches, title="Comunidades (nodos)",
           loc="lower right", bbox_to_anchor=(1.0, 0.0), frameon=True)

plt.tight_layout()
plt.show()
```

```
C:\Users\Bersd\AppData\Local\Temp\ipykernel_38920\70611225.py:17: MatplotlibDeprecationWarning: The get_
cmap function was deprecated in Matplotlib 3.7 and will be removed in 3.11. Use ``matplotlib.colormaps[n
ame]`` or ``matplotlib.colormaps.get_cmap()`` or ``pyplot.get_cmap()`` instead.
  base_cmap = plt.cm.get_cmap("tab20", max(K, 20))  # asegura ≥20 colores
```



Interpretación del subgrafo (N = 50)

Objetivo: analizar las principales comunidades detectadas dentro del subgrafo reducido a las **50 cuentas más conectadas** en el dataset `Tweets_Dataset`.
Esta reducción concentra la red en los **nodos más influyentes**, permitiendo observar las **interacciones clave** y los **núcleos temáticos** del grafo.

Comunidades detectadas

Cada comunidad agrupa usuarios que interactúan con mayor frecuencia entre sí.
En este ejemplo:

- **Comunidad 1:** núcleo corporativo y viajeros frecuentes (p.ej. `@americanair`, `@delta`).
- **Comunidad 2:** conversaciones operativas y logísticas (Southwest, aeropuertos).
- **Comunidad 3:** promociones o reseñas positivas ligadas a JetBlue.
- **Comunidad 4:** entusiastas y fans de Virgin America.
- **Comunidad 5:** quejas puntuales a Delta.
- **Comunidad 6:** interacción entre cuentas corporativas y medios.

Lectura interpretativa

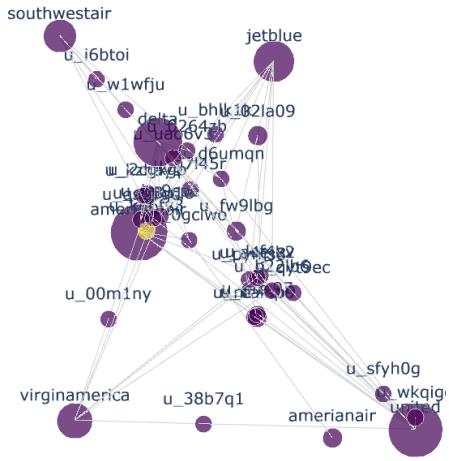
- **Núcleo central:** formado por aerolíneas principales (`@americanair`, `@united`), actuando como *hubs*.
- **Periferias diferenciadas:** JetBlue y Virgin America tienen grupos más independientes.
- **Subgrupos secundarios:** Delta y Southwest mantienen comunidades pequeñas conectadas al centro.
- **Modularidad (Q ≈ 0.3–0.4):** indica una **estructura temática clara** con relaciones cruzadas moderadas.

> En resumen: el subgrafo Top-50 revela los actores más influyentes y las agrupaciones naturales dentro de las conversaciones de aerolíneas, ideal para análisis de reputación o segmentación de

In [34]:
```
# ==========================================
# Visualización 3D del subgrafo (Plotly) — Tweets_Dataset
# ==========================================
import pandas as pd
import numpy as np
import re
import networkx as nx
import plotly.graph_objects as go

# ------- 1) Asegurar subgrafo H (top-50 por grado) -------
try:
    H
except NameError:
    # construir G (autor -> mencionado) con pesos por frecuencia
    df = pd.read_csv("Tweets_Dataset.csv")

    def extract_mentions(t):
        if pd.isna(t): return []
        return re.findall(r"@(\w+)", str(t))

    if "author" not in df.columns:
        df["author"] = df["name"].fillna("").astype(str) if "name" in df.columns else [f"user_{i}" for
    if "mentions" not in df.columns:
        df["mentions"] = df["text"].apply(extract_mentions)

    pairs = (
        df.explode("mentions")[["author","mentions"]]
          .dropna()
          .query("author.str.len()>0 and mentions.str.len()>0", engine="python")
    )
    pairs["mentions"] = pairs["mentions"].str.lower()
    pairs = pairs[pairs["author"].str.lower()!=pairs["mentions"]]

    w_edges = pairs.groupby(["author","mentions"]).size().reset_index(name="weight")

    G = nx.DiGraph()
    G.add_weighted_edges_from(w_edges[["author","mentions","weight"]].itertuples(index=False, name=None

    # top-50 por grado total
    deg_total = dict(G.degree())
    top_nodes = [n for n,_ in sorted(deg_total.items(), key=lambda kv: kv[1], reverse=True)[:50]]
    H = G.subgraph(top_nodes).copy()

    # componente principal (evitar nodos sueltos)
    if H.number_of_nodes() > 0:
        cc = max(nx.weakly_connected_components(H), key=len)
        H = H.subgraph(cc).copy()

# ------- 2) Comunidades (sobre no dirigido del subgrafo) -------
Gu = H.to_undirected()
comms = list(nx.algorithms.community.greedy_modularity_communities(Gu)) if Gu.number_of_edges()>0 else
comm_id = {}
for i, cset in enumerate(comms):
    for n in cset:
        comm_id[n] = i

# setear atributo 'community_id' en H
nx.set_node_attributes(H, {n: comm_id.get(n, -1)}, "community_id")

# tamaños por in-degree (popularidad)
in_deg = dict(H.in_degree())
vals = np.array(list(in_deg.values()), dtype=float) if in_deg else np.array([1.0])
rango = np.ptp(vals) if np.ptp(vals) > 0 else 1.0
sizes = (12 + (vals - vals.min())/rango * 28).tolist()  # 12..40 px

# ------- 3) Layout 3D y trazas -------
pos3d = nx.spring_layout(H, dim=3, seed=42)

x_nodes = [pos3d[n][0] for n in H.nodes()]
y_nodes = [pos3d[n][1] for n in H.nodes()]
z_nodes = [pos3d[n][2] for n in H.nodes()]

], []
```

```
for u, v in H.edges():
    x0, y0, z0 = pos3d[u]
    x1, y1, z1 = pos3d[v]
    edge_x += [x0, x1, None]
    edge_y += [y0, y1, None]
    edge_z += [z0, z1, None]

edge_trace = go.Scatter3d(
    x=edge_x, y=edge_y, z=edge_z,
    mode="lines",
    line=dict(color="lightgrey", width=1),
    hoverinfo="none",
    name="menciones"
)

node_colors = [H.nodes[n].get("community_id", 0) for n in H.nodes()]
node_labels = [str(n) for n in H.nodes()]

node_trace = go.Scatter3d(
    x=x_nodes, y=y_nodes, z=z_nodes,
    mode="markers+text",
    text=node_labels, textposition="top center", textfont=dict(size=10),
    marker=dict(
        size=sizes,
        color=node_colors,
        colorscale="Viridis",
        showscale=True,
        colorbar=dict(title="Comunidad")
    ),
    hoverinfo="text",
    name="usuarios"
)

fig = go.Figure(data=[edge_trace, node_trace])
fig.update_layout(
    title="Visualización 3D — Subgrafo top-50 por grado (tamaño ∝ in-degree)",
    showlegend=False,
    scene=dict(
        xaxis=dict(showbackground=False, visible=False),
        yaxis=dict(showbackground=False, visible=False),
        zaxis=dict(showbackground=False, visible=False)
    ),
    margin=dict(l=0, r=0, t=40, b=0)
)
fig.show()
```

Visualización 3D — Subgrafo top-50 por grado (tamaño ∝ in-degree)



Visualización 3D interactiva del subgrafo (Plotly)

Objetivo: explorar de forma **interactiva** la red de menciones en 3D, enfocándonos en el **subgrafo Top-N por grado** (usuarios más conectados).
Permite rotar/zoom/hover para inspeccionar **nodos, aristas, comunidades y etiquetas** de las cuentas más relevantes.

Qué hace la celda

1. **Construye/recupera** el grafo no dirigido `Gu` a partir de `Tweets_Dataset.csv` (autor → mencionado con **peso por frecuencia**).
2. Calcula **PageRank** (para escalar tamaño de nodo) y **comunidades** (Greedy Modularity) si no existen.
3. Selecciona el **subgrafo** con los **Top-N** nodos por grado y genera **posiciones 3D** (`spring_layout`).
4. Dibuja:
   - **Aristas** como segmentos 3D (color gris con opacidad).
   - **Nodos** con:
     - **Tamaño** ∝ PageRank (fallback: grado).
     - **Color** por **comunidad** (barra de color incluida).
     - **Etiquetas** para los `LABEL_K` nodos con mayor grado.
5. Configura una figura **Plotly** para navegación libre (pan/zoom/rotación) y **hover** con grado del nodo.

Parámetros útiles

- `N` (50–120): tamaño del subgrafo mostrado.
- `LABEL_K` : número de etiquetas visibles.
- `EDGE_W` , `EDGE_A` : grosor y opacidad de aristas.
- `NODE_MIN` , `NODE_MAX` : rango de tamaños de nodo.

Lectura rápida

PageRank (influencia).

- **Mismo color** ⇒ misma **comunidad**.
- **Muchos enlaces** ⇒ alta **conectividad** (hubs).

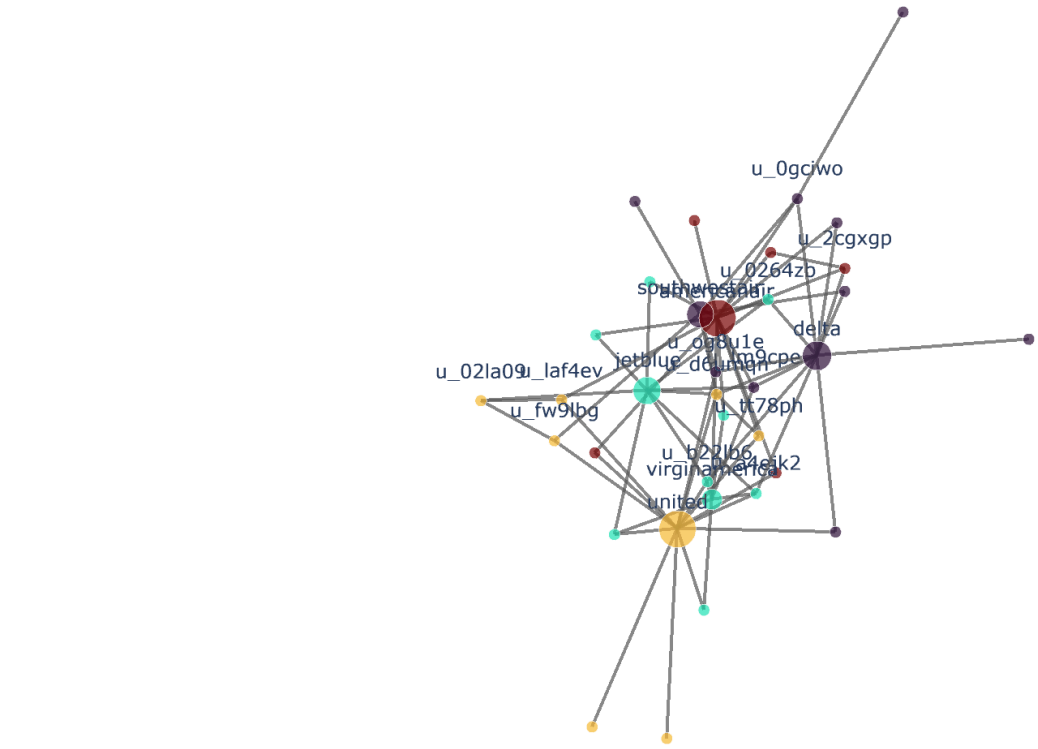
> Requisito: `pip install plotly`

In [35]:
```
# ===== Visualización 3D interactiva del subgrafo (Plotly) — Tweets_Dataset =====
# Requisitos:  pip install plotly

import pandas as pd
import numpy as np
import re
import networkx as nx
import plotly.graph_objects as go

# ---------- Parámetros ajustables ----------
N        = 50     # top-N por grado (50–120 recomendado)
SEED     = 42     # fija posiciones (reproducible)
LABEL_K  = 18     # cuántas etiquetas mostrar
EDGE_W   = 4      # grosor de aristas (2–7)
EDGE_A   = 0.70   # opacidad de aristas (0–1)
NODE_MIN = 8      # tamaño mínimo de nodo
NODE_MAX = 26     # tamaño máximo de nodo

# ---------- Asegurar Gu (grafo no dirigido) y pr/comms si no existen ----------
def extract_mentions(t):
    if pd.isna(t): return []
    return re.findall(r"@(\w+)", str(t))

try:
    Gu  # si ya existe desde pasos previos, lo reusamos
except NameError:
    # Construir G (autor -> mencionado) con pesos por frecuencia desde Tweets_Dataset.csv
    df = pd.read_csv("Tweets_Dataset.csv")
    if "author" not in df.columns:
        df["author"] = df["name"].fillna("").astype(str) if "name" in df.columns else [f"user_{i}" for
    if "mentions" not in df.columns:
        df["mentions"] = df["text"].apply(extract_mentions)

    pairs = (
        df.explode("mentions")[["author","mentions"]]
          .dropna()
          .query("author.str.len()>0 and mentions.str.len()>0", engine="python")
    )
    pairs["mentions"] = pairs["mentions"].str.lower()
    pairs = pairs[pairs["author"].str.lower() != pairs["mentions"]]

    w_edges = pairs.groupby(["author","mentions"]).size().reset_index(name="weight")

    G = nx.DiGraph()
    G.add_weighted_edges_from(w_edges[["author","mentions","weight"]].itertuples(index=False, name=None
    Gu = G.to_undirected()

# PageRank (si no existe) para escalado opcional del tamaño
try:
    pr
except NameError:
    pr = nx.pagerank(Gu, alpha=0.85) if Gu.number_of_nodes() else {}

# Comunidades (si no existen)
try:
    comms
except NameError:
    comms = list(nx.algorithms.community.greedy_modularity_communities(Gu)) if Gu.number_of_edges()>0 e

# ---------- Validaciones ----------
if Gu.number_of_nodes() == 0:
    raise ValueError("Grafo vacío.")

# ---------- 1) Subgrafo top-N por grado ----------
deg = dict(Gu.degree())
top_nodes = [n for n, _ in sorted(deg.items(), key=lambda kv: kv[1], reverse=True)[:N]]
```

```
# ---------- 2) Tamaño de nodo: PageRank si existe; si no, grado ----------
vals = np.array([pr.get(n, 0.0) for n in H.nodes()])
if vals.max() == 0:  # fallback a grado si pr=0
    vals = np.array([deg.get(n, 0) for n in H.nodes()], dtype=float)
vmax = vals.max() if vals.size and vals.max() > 0 else 1.0
node_sizes = NODE_MIN + (NODE_MAX - NODE_MIN) * (vals / vmax)

# ---------- 3) Color por comunidad ----------
node_comm = {}
for i, cset in enumerate(comms):
    for u in cset:
        node_comm[u] = i
color_vals = [node_comm.get(n, 0) for n in H.nodes()]  # numérico/categórico por comunidad

# ---------- 4) Posiciones 3D (layout con resorte) ----------
pos3d = nx.spring_layout(H, dim=3, seed=SEED)

x_nodes = [pos3d[n][0] for n in H.nodes()]
y_nodes = [pos3d[n][1] for n in H.nodes()]
z_nodes = [pos3d[n][2] for n in H.nodes()]

# Aristas como segmentos 3D (usar None para separar trazos)
ex, ey, ez = [], [], []
for u, v in H.edges():
    x0, y0, z0 = pos3d[u]
    x1, y1, z1 = pos3d[v]
    ex += [x0, x1, None]
    ey += [y0, y1, None]
    ez += [z0, z1, None]

# ---------- 5) Trazas Plotly ----------
EDGE_COLOR = f'rgba(60,60,60,{EDGE_A:.2f})'  # gris oscuro visible

edge_trace = go.Scatter3d(
    x=ex, y=ey, z=ez,
    mode='lines',
    line=dict(color=EDGE_COLOR, width=EDGE_W),
    hoverinfo='skip',
    showlegend=False
)

# Etiquetas: sólo para LABEL_K nodos con mayor grado
top_for_labels = sorted(H.degree, key=lambda kv: kv[1], reverse=True)[:LABEL_K]
label_set = {n for n, _ in top_for_labels}
texts = [str(n) if n in label_set else "" for n in H.nodes()]

node_trace = go.Scatter3d(
    x=x_nodes, y=y_nodes, z=z_nodes,
    mode='markers+text',
    text=texts, textposition='top center',
    marker=dict(
        size=node_sizes.tolist(),
        color=color_vals,          # por comunidad
        colorscale='Turbo',        # alterna 'Viridis' si prefieres
        showscale=True,
        colorbar=dict(title="Comunidad")
    ),
    hoverinfo='text',
    hovertext=[f"{n}<br>grado={deg.get(n,0)}" for n in H.nodes()],
    showlegend=False
)

# ---------- 6) Render (aristas primero, luego nodos) ----------
fig = go.Figure(data=[edge_trace, node_trace])
fig.update_layout(
    title=f"Red de menciones — 3D (subgrafo top-{N} por grado, nodos={H.number_of_nodes()}, aristas={H.
    width=1000, height=680,
    paper_bgcolor="white",
    scene=dict(
        xaxis=dict(showbackground=False, visible=False),
        yaxis=dict(showbackground=False, visible=False),
        zaxis=dict(showbackground=False, visible=False),
        aspectmode="data"
    )
)
```

```
    margin=dict(l=0, r=0, t=60, b=0)
)
fig.show()
```

Red de menciones — 3D (subgrafo top-50 por grado, nodos=35, aristas=68)



Visualización 2D interactiva (apta para daltónicos)

Objetivo: representar la red de menciones en **2D interactivo** con un estilo accesible para personas con **daltonismo**, manteniendo una visualización clara y nítida de las conexiones principales.

Qué hace la celda

1. **Construye (o reutiliza)** el grafo `Gu` (autor → mencionado) a partir del archivo `Tweets_Dataset.csv`, usando pesos por frecuencia.
2. Calcula **PageRank** y **comunidades** (Greedy Modularity) para agrupar usuarios similares.
3. Selecciona el **subgrafo Top-N por grado** (usuarios más conectados) y genera posiciones estables con `spring_layout`.
4. Dibuja:
   - **Aristas finas y translúcidas** para reducir ruido visual.
   - **Nodos** escalados por grado/PageRank.
   - **Colores y formas distintivas** por comunidad (paleta *Okabe–Ito* apta para daltónicos).
   - **Etiquetas** en los `LABEL_K` nodos más relevantes.
5. Muestra un gráfico **interactivo con Plotly**, donde puedes pasar el cursor para ver información de cada nodo y comunidad.

- `N` : número de nodos principales mostrados (50 por defecto).
- `LABEL_K` : cantidad de etiquetas visibles.
- `palette` : paleta cromática perceptualmente uniforme.
- `shapes` : distingue comunidades mediante diferentes formas.

Interpretación

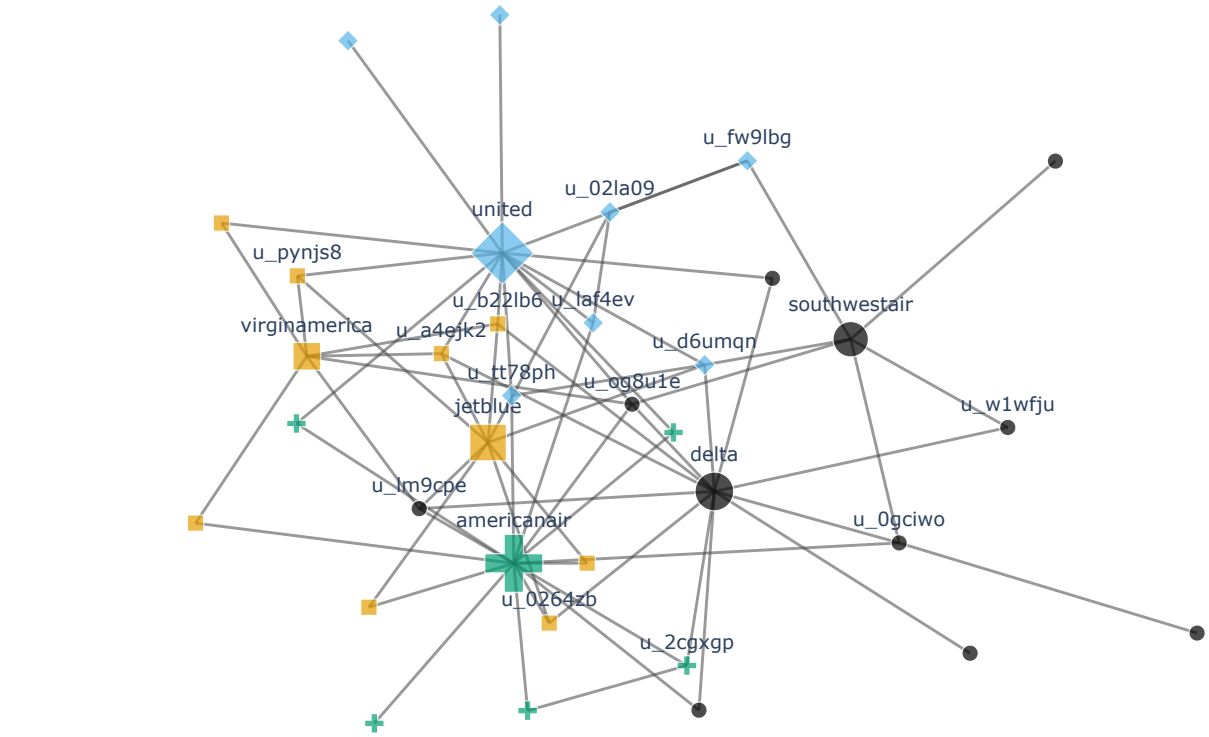
- **Colores y símbolos** → identifican comunidades distintas.
- **Tamaño del nodo** → refleja relevancia o influencia (PageRank).
- **Densidad de aristas** → muestra niveles de interacción.

> Ideal para informes o dashboards de análisis de redes sociales, con buena legibilidad en presentaciones o contextos accesibles.

In [36]:
```
# ===== Visualización 2D interactiva, aristas nítidas y apta para daltónicos — Tweets_Dataset =====
# Requisitos (una vez):  # !pip install plotly

import pandas as pd
import numpy as np
import re
import networkx as nx
import plotly.graph_objects as go
from collections import defaultdict

# ---------- Parámetros ----------
N = 50        # top-N por grado (solo para dibujar)
LABEL_K = 20  # cuántas etiquetas mostrar
SEED = 42

# ---------- Construir/red usar Gu (grafo no dirigido) ----------
def extract_mentions(t):
    if pd.isna(t): return []
    return re.findall(r"@(\w+)", str(t))

try:
    Gu  # si ya existe, úsalo
except NameError:
    # construir desde Tweets_Dataset.csv
    df = pd.read_csv("Tweets_Dataset.csv")
    if "author" not in df.columns:
        df["author"] = df["name"].fillna("").astype(str) if "name" in df.columns else [f"user_{i}" for
    if "mentions" not in df.columns:
        df["mentions"] = df["text"].apply(extract_mentions)

    pairs = (
        df.explode("mentions")[["author","mentions"]]
          .dropna()
          .query("author.str.len()>0 and mentions.str.len()>0", engine="python")
    )
    pairs["mentions"] = pairs["mentions"].str.lower()
    pairs = pairs[pairs["author"].str.lower() != pairs["mentions"]]

    w_edges = pairs.groupby(["author","mentions"]).size().reset_index(name="weight")

    G = nx.DiGraph()
    G.add_weighted_edges_from(w_edges[["author","mentions","weight"]].itertuples(index=False, name=None
    Gu = G.to_undirected()

# PageRank y comunidades si no existen
try:
    pr
except NameError:
    pr = nx.pagerank(Gu, alpha=0.85) if Gu.number_of_nodes() else {}

try:
    comms
except NameError:
    comms = list(nx.algorithms.community.greedy_modularity_communities(Gu)) if Gu.number_of_edges()>0 e

# ---------- Subgrafo top-N por grado ----------
```

```
deg = dict(Gu.degree())
top_nodes = [n for n, _ in sorted(deg.items(), key=lambda kv: kv[1], reverse=True)[:N]]
H = Gu.subgraph(top_nodes).copy()

# ---------- Posiciones ESTABLES ----------
pos_full = nx.spring_layout(Gu, seed=SEED)
pos = {n: pos_full[n] for n in H.nodes()}

# ---------- Aristas ----------
ex, ey = [], []
for u, v in H.edges():
    x0, y0 = pos[u]; x1, y1 = pos[v]
    ex += [x0, x1, None]; ey += [y0, y1, None]
edge_trace = go.Scatter(
    x=ex, y=ey, mode='lines',
    line=dict(color='rgba(70,70,70,0.55)', width=1.8),
    hoverinfo='skip', showlegend=False
)

# ---------- Tamaño de nodo: PageRank o grado ----------
val = np.array([pr.get(n, 0.0) for n in H.nodes()])
if val.max() == 0:
    val = np.array([deg.get(n, 0) for n in H.nodes()], dtype=float)
vmax = val.max() if val.size and val.max() > 0 else 1.0
sizes = 10 + 20 * (val / vmax)

# ---------- Comunidades (id por nodo) ----------
comm_id_of = {}
for i, cset in enumerate(comms, start=1):
    for u in cset:
        comm_id_of[u] = i

nodes_by_comm = defaultdict(list)
for n in H.nodes():
    nodes_by_comm[comm_id_of.get(n, 0)].append(n)

# Paleta Okabe-Ito (apta para daltónicos) + formas
palette = ['#000000','#E69F00','#56B4E9','#009E73','#F0E442','#0072B2','#D55E00','#CC79A7']
shapes  = ['circle','square','diamond','cross','x','triangle-up','triangle-down','star']

# Etiquetas: top LABEL_K por grado
label_nodes = {n for n, _ in sorted(H.degree, key=lambda kv: kv[1], reverse=True)[:LABEL_K]}

# ---------- Trazas por comunidad ----------
traces = [edge_trace]
node_order = list(H.nodes())  # para indexar sizes
for idx, (cid, nodes) in enumerate(sorted(nodes_by_comm.items(), key=lambda kv: len(kv[1]), reverse=Tru
    x = [pos[n][0] for n in nodes]; y = [pos[n][1] for n in nodes]
    text = [str(n) if n in label_nodes else "" for n in nodes]
    sizes_comm = [sizes[node_order.index(n)] for n in nodes]
    traces.append(go.Scatter(
        x=x, y=y, mode='markers+text',
        text=text, textposition='top center',
        marker=dict(
            size=sizes_comm,
            color=palette[idx % len(palette)],
            symbol=shapes[idx % len(shapes)],
            line=dict(color='white', width=0.8)
        ),
        name=f"Comunidad {cid} (n={len(nodes)})",
        hovertext=[f"{n}<br>grado={deg.get(n,0)}<br>comunidad={cid}" for n in nodes],
        hoverinfo='text'
    ))

# ---------- Render ----------
fig = go.Figure(data=traces)
fig.update_layout(
    title=f"Red de menciones — 2D interactivo (top-{N} por grado, nodos={H.number_of_nodes()}, aristas=
    width=980, height=680,
    xaxis=dict(visible=False), yaxis=dict(visible=False),
    paper_bgcolor='white', plot_bgcolor='white',
    legend=dict(title="Comunidades", orientation='v')
)
fig.show()
```

Red de menciones — 2D interactivo (top-50 por grado, nodos=35, aristas=68)



📊 Métricas clave sobre Tweets_Dataset (Red de menciones Twitter)

> Eje del análisis: red de menciones **autor → mencionado**
> (nodos = cuentas; aristas = menciones).

1️⃣ Degree Centrality (Grado)

Qué es: número de conexiones de un nodo.
En grafo dirigido:

- **In-degree:** cuántas veces te mencionan → **popularidad**.
- **Out-degree:** cuántas menciones haces → **actividad**.

Cálculo:

```
in_degree  = dict(G.in_degree())
out_degree = dict(G.out_degree())
```

Métricas clave para análisis de redes sociales (SNA + NLP)

Esta sección resume las **principales métricas de centralidad y cohesión** utilizadas para interpretar una red social
`mencionado`) dentro del dataset `Tweets_Dataset.csv`.

1. In-degree / Out-degree

- **In-degree alto**: usuarios o cuentas **muy mencionadas** → aerolíneas o figuras clave.
- **Out-degree alto**: usuarios **muy activos** → bots, soporte o community managers.

Interpretación:

- **In-degree alto** = relevancia reputacional.
- **Out-degree alto** = actividad y difusión (control de atención al cliente).

---

2. Betweenness Centrality

Evalúa cuántos **caminos más cortos** pasan por un nodo → mide **puentes entre comunidades**.
Ideal para detectar **nodos mediadores** (periodistas, líderes de opinión, viajeros frecuentes).

Interpretación:
Alta betweenness = usuarios que **controlan el flujo de información** entre grupos.

---

3. PageRank

Mide la **influencia global** considerando la calidad de las menciones.
Aerolineas mencionadas por usuarios influyentes tienden a mayor PageRank.

Interpretación:
Top PageRank = **influencers sistémicos** → embajadores o socios estratégicos.

---

4. Modularity (Q)

Cuantifica **qué tan bien se agrupan los nodos** en comunidades temáticas.
Valores de **Q ≈ 0.4–0.5** indican segmentación clara por marca o tipo de conversación.

Interpretación:
Cada comunidad = **segmento temático** (fans, quejas, marketing, soporte).

---

5. Density

Evalúa cuán **conectada globalmente** está la red (0–1).
En redes de menciones, suele ser baja porque los usuarios mencionan más a marcas que entre sí.

Interpretación:
Baja densidad = conversación **fragmentada** → fomentar interacciones laterales.

---

6. Clustering Coefficient

Mide la **cohesión local** (si los contactos de un nodo también se conectan entre sí).

Interpretación:

- **Clustering alto:** "tribus digitales" o comunidades fieles.
- **Clustering bajo:** usuarios aislados o menciones puntuales.

---

7. Sentimiento medio por comunidad

Combina **análisis de sentimientos** (NLP) con **estructura de red**.
Promedia `-1` (negativo) a `+1` (positivo) según tweets por comunidad.

- **< 0:** comunidades críticas (soporte o reclamos).
- **> 0:** comunidades promotoras o embajadoras.

Resumen ejecutivo

Métrica	Revela	Acción sugerida
In-degree / PageRank	Influencia y reputación	Monitorear o asociar
Betweenness	Puentes de información	Vigilar / aprovechar
Modularity	Segmentación temática	Mensajes diferenciados
Density / Clustering	Cohesión y actividad	Fomentar interacción lateral
Sentimiento por comunidad	Clima emocional	Soporte o fidelización focalizada

---

Recomendación final Incluye en el informe:

- Top-10 por **PageRank** y **Betweenness**
- Tabla de **comunidades** (tamaño + 3 cuentas clave + sentimiento)
- Visualización del **subgrafo top-N (2D o 3D)** para presentar la estructura global.

In [37]:
```
# ============================
# (Opcional) Tablas finales "para informe" — Tweets_Dataset
# ============================

# --- Asegurar que existen métricas ---
try:
    influ_table
except NameError:
    # Si no existe, recalculamos rápido las métricas clave
    pr = nx.pagerank(G, alpha=0.85)
    btw = nx.betweenness_centrality(G, normalized=True)
    indeg = dict(G.in_degree())
    outdeg = dict(G.out_degree())

    influ_table = pd.DataFrame({
        "node": list(G.nodes()),
        "pagerank": [pr.get(n, 0.0) for n in G.nodes()],
        "betweenness": [btw.get(n, 0.0) for n in G.nodes()],
        "in_degree": [indeg.get(n, 0) for n in G.nodes()],
        "out_degree": [outdeg.get(n, 0) for n in G.nodes()],
    }).sort_values(["pagerank","betweenness","in_degree"], ascending=False)

# --- Top-10 por PageRank ---
top_pr = (
    influ_table.sort_values("pagerank", ascending=False)
    .head(10)[["node","pagerank","in_degree","betweenness"]]
    .reset_index(drop=True)
)

print("Top-10 cuentas por PageRank (influencia global):")
display(top_pr)

# --- Top-10 por Betweenness ---
top_btw = (
    influ_table.sort_values("betweenness", ascending=False)
    .head(10)[["node","betweenness","in_degree","pagerank"]]
    .reset_index(drop=True)
)

print("Top-10 cuentas por Betweenness (conectores entre comunidades):")
display(top_btw)
```

```
Top-10 cuentas por PageRank (influencia global):
```

	node	pagerank	in_degree	betweenness
0	united	0.070167	1152	0.000000
1	americanair	0.069883	1151	0.000000
2	delta	0.048876	827	0.000000
3	jetblue	0.044565	768	0.000000
4	southwestair	0.041911	743	0.000000
5	virginamerica	0.025201	481	0.000000
6	uniited	0.000927	6	0.000000
7	mericanair	0.000746	7	0.000000
8	aericanair	0.000680	7	0.000000
9	u_g7l45r	0.000627	7	0.016953

```
Top-10 cuentas por Betweenness (conectores entre comunidades):
```

	node	betweenness	in_degree	pagerank
0	u_ne9rga	0.028373	5	0.000502
1	u_2yk0r2	0.025986	2	0.000454
2	u_mzotiw	0.023385	2	0.000271
3	u_j3tvwe	0.023302	4	0.000395
4	u_1oro16	0.021786	4	0.000357
5	u_6w3f2h	0.021440	1	0.000247
6	u_ues1i8	0.021031	3	0.000331
7	u_9ayqt4	0.019891	3	0.000314
8	u_0gciwo	0.019564	5	0.000509
9	u_xuegud	0.019077	3	0.000396

In [38]:
```
# ============================
# Comunidades: tamaño + sentimiento (formateado para informe)
# ============================
import pandas as pd
import numpy as np

# Asegurar tablas base
assert 'summary_comm' in globals(), "Falta summary_comm (tamaños por comunidad)."
assert 'comm_sent' in globals(), "Falta comm_sent (sentimiento por comunidad)."

comm_report = (
    summary_comm
    .merge(comm_sent, left_on="community_id", right_index=True, how="left")
    .fillna({"n_tweets": 0, "sent_mean": 0, "interpretación": "s/datos"})
    .assign(
        sent_mean=lambda d: d["sent_mean"].round(3),
        tasa_tweets=lambda d: (d["n_tweets"] / d["size"]).replace([np.inf, np.nan], 0).round(3)
    )
    .sort_values(["size", "n_tweets"], ascending=False)
    .reset_index(drop=True)
)

print("Comunidades: tamaño + sentimiento")
display(comm_report)

# (opcional) exportar
# comm_report.to_csv("Reporte_Comunidades.csv", index=False)
```

```
Comunidades: tamaño + sentimiento
```

	community_id	size	n_tweets	sent_mean	interpretación	tasa_tweets
0	1	337	1019	-0.294	negativo	3.024
1	2	332	976	-0.254	negativo	2.940
2	3	318	956	-0.275	negativo	3.006
3	4	303	856	-0.279	negativo	2.825
4	5	298	865	-0.299	negativo	2.903
5	6	239	699	-0.285	negativo	2.925
6	7	92	232	-0.194	negativo	2.522
7	8	61	141	-0.305	negativo	2.311
8	9	60	152	-0.283	negativo	2.533
9	10	54	149	-0.275	negativo	2.759
10	11	52	129	-0.225	negativo	2.481
11	12	51	109	-0.156	negativo	2.137
12	13	48	117	-0.231	negativo	2.438
13	14	44	94	-0.298	negativo	2.136
14	15	40	88	-0.148	negativo	2.200
15	16	35	100	-0.310	negativo	2.857
16	17	32	86	-0.186	negativo	2.688
17	18	30	81	-0.296	negativo	2.700
18	19	27	65	-0.277	negativo	2.407
19	20	26	77	-0.273	negativo	2.962
20	21	5	9	-0.111	negativo	1.800